Multitasking on the Cray-2 and Cray Y-MP: An Experimental Study

R. A. Fatoohi [1]

Report RNR-88-001, December 1988

Sterling Software
NASA Ames Research Center
Moffett Field, CA 94035, USA

Abstract. This paper presents the results of an experiment to study the performance of multitasking techniques on the Cray-2 and Cray Y-MP using a single algorithm. The algorithm is a compact difference scheme for the solution of the incompressible, two-dimensional, time dependent Navier-Stokes equations. Two implementations of multitasking on both machines are considered. These are: macrotasking (parallelism at the subroutine level) and microtasking (parallelism at the do-loop level). These two techniques are briefly described. The implementation of the algorithm is discussed in relation to these techniques and the results for three problem sizes are presented. The timing results for both techniques using few processors are in general comparable on both machines. For the eight processor case on the Cray Y-MP, microtasking outperformed macrotasking for the small problems while the latter outperformed the former for the larger problem. The best achieved processing rate on the Cray Y-MP is 1.2 GFLOPS using macrotasking. Finally comparisons are made and conclusions are drawn.

---

[1] This research was supported by NASA Contract No. NAS2-11555 while the author was an employee of Sterling Software under contract to the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center.

1. Introduction

It is now clear that parallelism is the road to increased supercomputer performance. However, it is not clear whether this parallelism should be by way of a relatively few powerful processors in an MIMD organization or many much less powerful processors in a SIMD organization. Both approaches have their advocates.

There have been a substantial number of theoretical studies of the performance of algorithms on parallel supercomputers but far fewer actual experimental studies. Insofar as MIMD supercomputers are concerned, there are even fewer actual studies. Although multitasking has been available on the Cray series of multiprocessor supercomputers since 1984, its use has been very limited. The exceptions are some experiments on the Cray X-MP and Cray-2. Some results of macrotasking (previously called multitasking) on the Cray X-MP family were reported by Larson [9], Chen et al. [3], Seager [12], and Mandell [10]. Larson reported speedups of 1.86 to 1.89 on the Cray X-MP/2 for a particle-in-cell code, a weather forecasting code, and a three dimensional seismic migration code. Chen et al. used two sets of multitasking routines on the Cray X-MP/2 to study the solution of symmetric linear systems using LU decomposition and Cholesky factorization algorithms. They reported speedups of 0.76 to 1.80 by using the standard multitasking routines and speedups of 1.39 to 1.92 by using a set of assembly language (CAL) routines. Seager reported speedups of 1.54 to 1.95 on the Cray X-MP/2 and speedups of 2.69 to 2.90 on the Cray X-MP/4 for the solution of symmetric linear systems using three conjugate gradient methods. Mandell reported a speedup of 2.92 on the Cray X-MP/4 for a one dimensional hydrodynamics code. Results of microtasking were first reported by Misegades et al. [11] where they achieved speedups of 3.40 and 3.65 on the Cray X-MP/4 for two CFD codes. Bieterman [2] reported speedups of 1.22 to 2.36 on the Cray X-MP/4 using microtasking for the partial differential equation package PLTMG. On the Cray-2, Swisshelm [13] reported a speedup of 2.67 using macrotasking for a three dimensional Navier-Stokes algorithm. These experiments show that reasonable speedups can be achieved on these systems although not in all cases. Clearly there is a need for more experiments in this field.

The work reported here was just such an experiment. A two dimensional Navier-Stokes algorithm was implemented on the Cray-2 and Cray Y-MP using two multitasking techniques: macrotasking and microtasking. Both techniques are briefly described in section 2. The algorithm is briefly described in section 3. The implementation of the algorithm using macrotasking and microtasking is described in section 4. Section 5 contains the results of the implementation. Finally, section 6 contains a comparison of performance of both techniques on both machines and some concluding remarks.

2. The Cray-2 and Cray Y-MP Computer Systems

2.1. Hardware Overview

The Cray-2 is an MIMD supercomputer with four CPUs, a foreground processor which controls I/O, and a main memory. The main memory has 256 million 64 bit words organized in four quadrants of 32 banks each. Each CPU has access to one quadrant during each clock cycle. The clock cycle is 4.1 nanoseconds. The results reported here were obtained using the new Cray-2 with a shorter main memory access time (80 ns DRAM) at NASA Ames Research Center.

The Cray Y-MP architecture is an evolutionary step from the Cray X-MP series of computers. It has eight CPUs, 32 million 64 bit words of main memory, and 256 million words of SSD memory. The main memory is organized in four sections of 64 banks each. The first Cray Y-MP was delivered to NASA Ames Research Center last August with a 6.3 nanosecond clock cycle. This work was performed using this machine during the acceptance period.

2.2. Software Support

Both machines run the UNICOS operating system which is based on UNIX System V. Under UNICOS, the four processors can operate independently on separate jobs or concurrently on a single problem. The former mode is called multiprogramming while the latter is called multitasking. Multiprogramming is defined as a property of the operating system that permits overlapping and interleaving the execution of more than one program. On both machines, individual processors, scheduled in a multiprogramming mode, are treated as additional resources to be allocated to jobs. As a result, total system throughput increases, but a single job will not see any special gain. Multitasking is defined as the structuring of a program into multiple parts that can execute concurrently. Performance gains result when these parts are run at the same time on different processors of a multiprocessing system. Currently there are two implementations of multitasking: macrotasking and microtasking.

2.3. Macrotasking

Macrotasking is the process of partitioning a program into two or more tasks at the subroutine level. The granularity of these tasks may be large. The system software for both machines provide a library of Fortran-callable subroutines that implements a basic set of primitive macrotasking functions [4]. These subroutines are called by a macrotasked program as required to create and synchronize task execution. Among these subroutines are: "tskstart(t,s)," to initiate a task with identification t and subroutine s; "tskwait(t)," to wait for the task t to complete execution; "lockasgn(l)," to identify the integer variable l as a lock; "lockon(l)," to set the lock l; and "lockoff(l)," to clear the lock l.

In addition, the Cray Fortran extensions for multitasking include a new COMMON statement termed TASK COMMON. Data that is shared between subroutines but which is local to a task should be kept in TASK COMMON blocks.

2.4. Microtasking

Microtasking is the process of partitioning a program into parts at the do-loop level. The granularity of these parts may be small. Microtasking is specified by a number of user-supplied directives that appear as comment lines [4]. A preprocessor, called premult, interprets the directives and then rewrites the code to make microtasking library calls. The addition of the microtasking directives does not reduce the portability of the code. Among these directives are: "getcpus n," to specify the number of processors permitted to work on the program; "micro," to designate a subroutine to be microtasked; "doglobal," to designate the start of a do loop whose individual iterations can be done in parallel; "process," to mark the beginning of a block of code which can be executed by only one processor; "alsoprocess," to mark the end of one process and the beginning of another; "endprocess," to mark the end of a process; and "relcpus," to release all the processors except one.

The portions of the code following doglobal and process primitives, where multitasking can be done, are called control structures. There is an implicit synchronization point at the bottom of every control structure. Unlike vectorization, the scope of the variables used in the program should be given special consideration. Global variables, like in common blocks or in a subroutine's argument list, may be modified only inside a control structure. Also, local variables defined in a control structure cannot be relied on outside the control structure.

3. The numerical algorithm

The Navier-Stokes equations for the two-dimensional, time dependent flow of a viscous incompressible fluid may be written, in dimensionless variables, as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{3.1}$$

$$\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = \zeta, \tag{3.2}$$

$$\frac{\partial \zeta}{\partial t} + \frac{\partial}{\partial x}(u\ \zeta) + \frac{\partial}{\partial y}(v\ \zeta) = \frac{1}{\mathrm{Re}}\nabla^2\ \zeta, \tag{3.3}$$

where $\vec{u} = (u,v)$ is the velocity, ζ is the vorticity and Re is the Reynolds number.

Consider the problem of approximating the solution of equations (3.1) to (3.3) in the square domain $0 \le x \le 1$, $0 \le y \le 1$ with the boundary conditions

$u = 1$ and $v = 0$ at $y = 1$ and $u = v = 0$ elsewhere. The numerical method used to approximate these equations is based on the compact differencing schemes which require the use of only the values of the dependent variables in and on the boundaries of a single computational cell [8]. Fatoohi and Grosch [7] used this method to solve these equations on the MPP, Flex/32 and one processor of the Cray-2. This is done as follows: subdivide the computational domain into rectangular cells. The center of a cell is at $(i+1/2, j+1/2)$. Apply the centered difference operator to equations (3.1) and (3.2), which yields

$$\delta_x U_{i+1/2,j+1/2} + \delta_y V_{i+1/2,j+1/2} = 0, \tag{3.4}$$

$$\delta_x V_{i+1/2,j+1/2} - \delta_y U_{i+1/2,j+1/2} = \zeta_{i+1/2,j+1/2}. \tag{3.5}$$

The adaptation of this algorithm to parallel computers can be simplified by the introduction of box variables to represent the velocity field. The box variables are defined at the corners of the cells so that the average of two adjacent box variables is equal to the velocity field on the included side. The set of difference equations and boundary conditions in terms of the box variables are solved using a cell relaxation scheme which is equivalent to an SOR method [8]. The compact difference approximation to equation (3.3) results in an implicit set of equations which are solved by an ADI method [6].

In brief, the solution procedure for the Navier-Stokes equations is as follows: with an assumed distribution of ζ and one component of $\vec{u}$ prescribed on the boundary, equations (3.1) and (3.2) are solved by the cell relaxation method. A new distribution of ζ is then determined by solving equation (3.3) by the ADI method using boundary conditions for ζ which are implied by the other component of $\vec{u}$ prescribed on the boundary. A repetition of the above process then yields the velocity and vorticity at any later time.

4. Implementation

4.1. Macrotasking

The Navier-Stokes algorithm was macrotasked on the Cray-2 and Cray Y-MP for domains of sizes $n \times n$ grid points with $n = 64$, 128 and 256 and using p processors where $p = 2$, 4 or 8, for the Cray Y-MP. Also, the serial version of the code was run on one processor of both machines for comparison. The parallel implementation of the algorithm was done as follows. First, the domain was decomposed vertically into n by n/p strips for implementing the relaxation scheme, computing parameters and setting boundary conditions, and solving tridiagonal systems distributed over columns. Then the domain was decomposed horizontally into n/p by n strips for solving tridiagonal systems distributed over rows. The codes were written in Fortran augmented with subroutine calls to the macrotasking library. Each strip was assigned to a task. The main program, or initial task, initialized control variables for tasks and locks, started the processing tasks, waited for them to complete, and performed the input and output operations. The cell relaxation method was implemented by using a four color reordering scheme, see [5] for details. The two sets of the tridiagonal systems were solved by the Gaussian elimination algorithm for all systems of each set in parallel [6]. The inner loops of all codes were fully vectorized.

In order to satisfy data dependencies between segments of the code running many tasks, barriers were used. At a barrier, tasks must wait until every task running the code arrives at that point. After all have arrived, one task executes a section of sequential code, which may be null, after which all tasks exit the barrier. The two lock barrier [1] was used in this work. This barrier was implemented with two locks and a shared counter. Each time step required four barriers for the ADI method, three barriers for computing parameters and setting boundary conditions, and two barriers for each iteration of the relaxation scheme.

4.2. Microtasking

The Navier-Stokes algorithm was microtasked on the Cray-2 and Cray Y-MP for the three domain sizes, as above, using p processors where $p = 2$, 3 or 4 for the Cray-2 and $p = 2$, 3, 4, 5, 6, 7 or 8 for the Cray Y-MP. The implementation was quite simple. All steps of the solution procedure were microtasked; this includes the routines that take very small percentage of the total execution time. A set of microtasking directives were inserted in the serial version of the program. A total of 39 directives for the 64×64 problem and 55 directives for the 128×128 and 256×256 problems were added to the program. No changes in data scoping of variables were required, since mostly global variables were used. It is worth mentioning that the codes remained portable after inserting the microtasking directives. These microtasked codes can run on one processor of these machines as well as on any serial machine.

5. Results and Discussion

5.1. Macrotasking

The performance of the multitasked algorithms on both machines was evaluated by using speedup and efficiency measures. Speedup was computed by taking the ratio of the time to solve the problem using one processor to the time to solve the same problem using p processors. Efficiency was determined by taking the ratio of the speedup using p processors to p. Table I and II contain the measured execution time during dedicated time, the processing rate, the speedup and the efficiency for the 64 × 64, 128 × 128 and 256 × 256 problems using macrotasking on the Cray-2 and Cray Y-MP, respectively. The execution time represents the total time of all steps of the solution procedure, excluding I/O, for ten time steps. The processing rate is computed by counting the additions, multiplications and divisions only; division is counted as a single operation. The total number of arithmetic operations for the multitasking cases is the same as for the one processor case; no additional arithmetic operations are involved in the overhead.

Number of processors	Execution time (seconds)	Processing Rate (MFLOPS)	Speedup	Efficiency (%)
		64 × 64 grid points		
1	2.589	113	-	-
2	1.695	173	1.53	76.4
4	1.087	269	2.38	59.5
		128 × 128 grid points		
1	24.963	117	-	-
2	13.981	209	1.79	89.3
4	7.837	374	3.19	79.6
		256 × 256 grid points		
1	231.548	117	-	-
2	119.572	226	1.94	96.8
4	63.924	422	3.62	90.6

Table I. The execution time in a dedicated environment, processing rate, speedup, and efficiency for the Navier-Stokes Algorithm on the Cray-2 using macrotasking.

Number of processors	Execution time (seconds)	Processing Rate (MFLOPS)	Speedup	Efficiency (%)
		64 × 64 grid points		
1	1.643	178	-	-
2	0.990	296	1.66	83.0
4	0.674	434	2.44	60.9
8	0.948	309	1.73	21.7
		128 × 128 grid points		
1	15.443	189	-	-
2	8.427	347	1.83	91.6
4	5.101	574	3.03	75.7
8	5.001	585	3.09	38.6
		256 × 256 grid points		
1	142.661	189	-	-
2	74.166	364	1.92	96.2
4	38.558	700	3.70	92.5
8	22.422	1203	6.36	79.5

Table II. The execution time in a dedicated environment, processing rate, speedup, and efficiency for the Navier-Stokes Algorithm on the Cray Y-MP using macrotasking.

For the one processor case, the processing rate on the Cray-2 ranges between 113 and 117 MFLOPS. The latter represents about 24% of the peak performance rate of a single processor of the Cray-2 (488 MFLOPS). The processing rate on the Cray Y-MP for the one processor case ranges between 178 and 189 MFLOPS. The latter represents about 60% of the peak performance rate of a single processor of the Cray Y-MP (317 MFLOPS). This means that one processor of the Cray Y-MP outperformed one processor of the Cray-2 by about 60% for this algorithm. The slight improvement in the processing rate for the larger problems on both machines is due to the fact that the 64 × 64 problem has many short vectors while the other two problems have only long vectors.

The highest obtained processing rate for the macrotasked algorithm on the Cray-2 was 422 MFLOPS. This is about 22% of the peak performance rate of the Cray-2 (1951 MFLOPS). The highest obtained processing rate for the same algorithm on the Cray Y-MP was 1203 MFLOPS. This is about 47% of the peak performance rate of the Cray Y-MP (2540 MFLOPS). This means the Cray Y-MP outperformed the Cray-2 by a factor of 2.85 for this algorithm using macrotasking.

The results listed in Tables I and II present the best case times out of many runs in a dedicated environment. There is a discrepancy between the timing results of the same job especially for small jobs (jobs on the order of seconds) due to a number of factors. Among these factors are: the time to start a task, which may take somewhere between few milliseconds up to few seconds for eight tasks, and other activities on the system even in a dedicated environment. These differences range between 50% for small jobs to less 1% for big jobs (jobs on the order of minutes).

As the number of processors in use was increased, the computation cost per processor decreased, while the overhead cost increased. This caused a degradation in the efficiency of the macrotasked algorithm for eight processors on the Cray Y-MP, especially for the small problems where no gain was achieved beyond four processors. Increasing the number of the grid points caused an increase by the same ratio in the computation cost and no change in the synchronization and task starting costs. This resulted in the improvement on the performance of the algorithm for the 256 $\times$ 256 problem.

5.2. Microtasking

Table III and IV contain the measured execution time, the processing rate, the speedup and the efficiency for the 64×64, 128×128 and 256×256 problems on dedicated time using microtasking on the Cray-2 and Cray Y-MP, respectively. These parameters are computed as in section 5.1. A variation in the timing results was also noticeable here for small jobs. The highest obtained processing rate on the Cray-2 was 387 MFLOPS. This is about 20% of the peak performance rate of the Cray-2. The highest obtained processing rate on the Cray Y-MP was 766 MFLOPS. This is about 30% of the peak performance rate of the Cray Y-MP. This means that the Cray Y-MP outperformed the Cray-2 by a factor of about 2 for this algorithm using microtasking.

Number of processors	Execution time (seconds)	Processing Rate (MFLOPS)	Speedup	Efficiency (%)
		64×64 grid points		
1	2.589	113	-	-
2	1.622	180	1.60	79.8
3	1.250	234	2.07	69.0
4	0.952	308	2.72	68.0
		128×128 grid points		
1	24.963	117	-	-
2	14.240	206	1.75	87.7
3	10.103	290	2.47	82.4
4	8.086	362	3.09	77.2
		256×256 grid points		
1	231.548	117	-	-
2	125.079	216	1.85	92.6
3	88.208	306	2.63	87.5
4	69.663	387	3.32	83.1

Table III. The execution time in a dedicated environment, processing rate, speedup, and efficiency for the Navier-Stokes Algorithm on the Cray-2 using microtasking.

Number of processors	Execution time (seconds)	Processing Rate (MFLOPS)	Speedup	Efficiency (%)
		64 × 64 grid points		
1	1.643	178	-	-
2	1.022	286	1.61	80.4
3	0.778	376	2.11	70.4
4	0.655	447	2.51	62.7
5	0.593	494	2.77	55.4
6	0.546	536	3.01	50.2
7	0.524	559	3.14	44.8
8	0.502	583	3.27	40.9
		128 × 128 grid points		
1	15.443	189	-	-
2	9.168	319	1.68	84.2
3	6.904	424	2.24	74.6
4	5.723	511	2.70	67.5
5	5.087	575	3.04	60.7
6	4.654	629	3.32	55.3
7	4.315	678	3.58	51.1
8	4.088	716	3.78	47.2
		256 × 256 grid points		
1	142.661	189	-	-
2	81.783	330	1.74	87.2
3	61.029	442	2.34	77.9
4	50.459	535	2.83	70.7
5	44.523	606	3.20	64.1
6	40.568	665	3.52	58.6
7	37.724	715	3.78	54.0
8	35.224	766	4.05	50.6

Table IV. The execution time in a dedicated environment, processing rate, speedup, and efficiency for the Navier-Stokes Algorithm on the Cray Y-MP using microtasking.

The results listed in Table IV shows a steady improvement on the performance of the Cray Y-MP when the number of processors in use was increased, even for the small problem. This is because the granularity of the tasks is independent on the number of processors used for microtasking.

6. Comparisons and Concluding Remarks

The Cray Y-MP, based on its peak performance rate, is faster than the Cray-2 by only 30%. Moreover, each processor of the Cray-2 is supposed to be faster than each processor of the Cray Y-MP by more than 50% (6.3/4.1). However, the results listed in Tables I through IV showed that the Cray Y-MP outperformed the Cray-2 for this algorithm by factors of 1.6, 2.85, and 2 using one processor, macrotasking, and microtasking respectively. This can be attributed to the slower main memory performance on the Cray-2. More specifically, there is only one path to main memory, and the DRAM chips in main memory are quite slow. The obtained speedups on both machines using few processors are in general comparable. The additional four processors on the Cray Y-MP, compared to the Cray-2, helped in achieving better performance for this algorithm. This was significant for the 256 $\times$ 256 problem using macrotasking where a processing rate of 1.2 GFLOPS was achieved, compared to 700 MFLOPS for the four processor case. However, for the smaller problems using macrotasking and for microtasking the improvement in the performance of the Cray Y-MP using more than four processors was less significant.

As shown in Tables I through IV, the macrotasked algorithm outperformed the microtasked algorithm for the 256 $\times$ 256 problem on both machines while the latter outperformed the former for the smaller problems using eight processors. The results for the other cases were in general comparable. The differences between both techniques are more significant on the Cray Y-MP than on the Cray-2. Microtasking outperformed microtasking on the Cray Y-MP by about 89%, for the 64 $\times$ 64 problem using eight processors, while macrotasking outperformed microtasking by about 57%, for the 256×256 problem using eight processors. These differences can be attributed to the granularity of the tasks and the overheads involved in synchronizing these tasks. Both techniques proved to be effective in multitasking the algorithm. The major attributes of microtasking are: ease of implementation, portability, and better performance for small granularity. On the other hand, macrotasking provides more control over synchronization and better performance for large granularity. Macrotasking requires considerable effort for controlling and synchronizing the tasks. Microtasking, on the other hand, requires less effort and is similar to vectorization except that special attention should be made to the scope of the variables used in the program. Both macrotasking and microtasking, like vectorization, may require a restructuring of the algorithm to achieve better performance.

These results and analysis reported here were for a single algorithm. More experiments may be needed to provide a better understanding of the different factors influence the performance of these techniques on both machines. A new multitasking technique, called autotasking, will also be exploited and the results will be compared with the results reported here.

References

[1] Axelrod, T. S., "Effects of Synchronization Barriers on Multiprocessor Performance," J. Parallel Computing, 3, 1986, 129-140.

[2] Bieterman, M., "Microtasking General Purpose Partial Differential Equation Software on the CRAY X-MP," Report ETA-TR-68, Boeing Computer Services, 1987.

[3] Chen, S. S., Dongarra, J. J., and Hsiung, C. C., "Multiprocessing Linear Algebra Algorithms on the CRAY X-MP-2: Experiences with Small Granularity," J. Parallel & Distributed Computing, 1, 1984, pp. 22-31.

[4] Cray Research, Inc., "Cray-2 Multitasking Programmer's Manual," Publication SN-2026 B, June 1988.

[5] Fatoohi, R. A. and Grosch, C. E., "Implementation of a Four Color Cell Relaxation Scheme on the MPP, Flex/32 and Cray/2," Proc. 1987 Int. Conf. Parallel Processing, pp. 424-426.

[6] Fatoohi, R. A. and Grosch, C. E., "Implementation of an ADI Method on Parallel Computers," J. Scientific Computing, Vol. 2, No. 2, 1987, pp. 175-193.

[7] Fatoohi, R. A. and Grosch, C. E., "Implementation and Analysis of a Navier-Stokes Algorithm on Parallel Computers," Proc. 1988 Int. Conf. Parallel Processing, Vol. III, pp. 235-242.

[8] Gatski, T. B., Grosch, C. E., and Rose, M. E., "A Numerical Study of the Two-Dimensional Navier-Stokes Equations in Vorticity-Velocity Variables," J. Comput. Phys., Vol. 48, No. 1, 1982, pp. 1-22.